CHAPTER 8
CONCLUSIONS

8.1. RECAPITULATION

In surveying the field of visual programming, we looked first at diagrams, and made the distinction between metric, topological, and symbolic realms. Within the topological realm, we identified three conventions used in diagramming software systems - adjoinment, linking, and containment. We then surveyed the use of diagrams in computer science, followed by the use of diagrams in visual programming systems.

In the following three chapters we produced prototypical visual programming systems. The first system operated on top of the Unix Shell, *awk*, and SASL. The second system operated over APL. The third system generated Mathematica code.

We then considered the issues of system design, and looked at the work of Buhr in depth. We generated new conventions as extensions of Buhr's notation to be used in representing Ada 9X concepts.

In a chapter entitled *The Limits of the Visual* we examined existing textual complexity metrics and considered their translation for use on visual programs. We introduced several new metrics, the primary one being Graphic Token Count. Using Textual Token Count and Graphic Token Count as tools, we analyzed tree and graph representations and concluded that textual

representation is more efficient than graphic representation, except for graphs, where graphic illustrations are equally compact and much more understandable. We then looked at examples from the previous chapters in light of these complexity measures.

8.2. CONCLUSIONS

From the last chapter, it is clear that fully general fully visual programming languages are not practical. The symbolic realm allows for much more compact representation of algorithms. And the symbolic realm has the power of naming, which allows self-referentiality and recursion.

So our first strong conclusion is that research into fully general visual programming languages will prove fruitless, as measured by the ability of such a language to be used to build large systems with productivity matching that of textual languages.

Much of the attraction of current visual models is their domain-specificity - they are used to build small programs on top of large libraries of pre-existing routines written in textual languages. For this activity, the visual realm will work well, as has been demonstrated here in the development of several such dialects. So visual shells and visual glue programs that allow routines to be strapped together will continue to proliferate. Yet they will only work on top of routines that will be written in textual languages.

Our second strong conclusion, is that the visual is more effective in the representation of graphs than the symbolic is, as measured by a textual and graphic token count metric. As a corrolary, it makes most sense to use the graphic for graph representation rather than tree representation - trees can be expressed well textually, but graphs cannot.

This suggests that the most fruitful use of the visual will be in those aspects of programming activity that are graph-intensive. The first area we can identify is the modeling of data, a field that is already permeated with graphic tools. The second is in the area of system design, where attention is on processes and places, not on memory locations and mathematical expressions.

Another strong conclusion is that the concept of the H-graph is very much at the root of any successful large-scale visual system. Without this abstraction capability, graphic representations become non-planar and muddled, a result of the hard constraints of planarity and resolution.

Our analysis of computer science diagrams revealed that the three conventions of adjoinment, linking, and containment can be combined to generate all the types of diagrams currently seen. This suggests that in the field of CASE tool generators, or meta-CASE tools, that such conventions, combined with H-graph capability, can be used and combined to create any conceivable type of model.

Finally, the work of Tufte on data density suggests that much of the power of the visual comes in the metric, not the topological domain. Without the restriction of textual labeling, and with the power of a continuous scale, many points of information can be organized and presented for simultaneous viewing. The Balsa system is a case in point of the power of using metric space as a way of gaining insight into computer programs.

Metric space can have a time axis. The use of the visual for scoring is powerful, as can be scene in the work of Buhr. Other examples of scoring occur in the use of computers to script video and audio, and these uses of the visual, rooted in metric time, will prove fruitful.

It is obvious that many programming activities lie strongly in the symbolic realm, and make use of skills that of all our senses are closest to the auditory. It is also evident from the history of computing literature and from the proliferation of CASE tools and visualization tools that there is a portion of programming activities that lend themselves to visual treatment. Already our programming tools are moving toward a hybrid state, where at a particular stage in a project we move from a design tool into source coding. A fruitful area of research is in mapping the graphic and textual pieces that can be combined into a productive programming environment. The work of Buhr, in which high-level graph-like design activities are handled visually, and lower level, expression-intense logic is handled textually is the best current hypothesis about how the two modes can fit together. It is likely that instead of visualizers that produce source, we will eventually work with languages that are hybrids, allowing both the power of naming and the power of diagrams to be used in programming.